Field Extension in Secret-Shared Form and Its Applications to Efficient Secure Computation

Ryo Kikuchi[1], Nuttapong Attrapadung[2], Koki Hamada[1], Dai Ikarashi[1], Ai Ishida[2], Takahiro Matsuda[2], Yusuke Sakai[2], and Jacob C. N. Schuldt[2]

[1] NTT,
kikuchi_ryo@fw.ipsj.or.jp, {koki.hamada.rb, dai.ikarashi.rd}@hco.ntt.co.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST),
{n.attrapadung, a.ishida, t-matsuda, yusuke.sakai, jacob.schuldt}@aist.go.jp

Abstract. Secure computation enables participating parties to jointly compute a function over their inputs while keeping them private. Secret sharing plays an important role for maintaining privacy during the computation. In most schemes, secret sharing over the *same* finite field is normally utilized throughout all the steps in the secure computation. A major drawback of this “uniform” approach is that one has to set the size of the field to be as large as the maximum of all the lower bounds derived from all the steps in the protocol. This easily leads to a requirement for using a large field which, in turn, makes the protocol inefficient. In this paper, we propose a “non-uniform” approach: dynamically changing the fields so that they are suitable for each step of computation. At the core of our approach is a surprisingly simple method to extend the underlying field of a secret sharing scheme, in a non-interactive manner, while maintaining the secret being shared. Using our approach, default computations can hence be done in a small field, which allows better efficiency, while one would extend to a larger field only at the necessary steps. As the main application of our technique, we show an improvement upon the recent actively secure protocol proposed by Chida et al. (Crypto’18). The improved protocol can handle a binary field, which enables XOR-free computation of a boolean circuit. Other applications include efficient (batch) equality check and consistency check protocols, which are useful for, e.g., password-based threshold authentication.

Keywords: secure computation, secret sharing, active security

1 Introduction

Secret-sharing-based secure computation enables parties to compute a function of a given set of inputs while keeping these secret. The inputs are distributed to several parties via a secret sharing scheme, and the parties then compute the function by interacting with each other. Throughout the above steps, any information except the output must be kept secret to the parties.

Secure computation should satisfy the security notions, such as privacy and correctness, in the presence of an adversary, which might compromise some of the parties participating in the computation. There are two classical adversary models capturing different adversarial behaviors: passive (i.e., semi-honest) and active (i.e., malicious). The latter provides a stronger security guarantee as an actively secure protocol will remain secure in the presence of an adversary following an arbitrary adversarial strategy. Another metric of security is the number of parties that an adversary can corrupt. The setting in which an adversary is allowed to corrupt up to half of the parties, is referred to as honest majority. Unconditionally secure protocols can only be realized in the honest majority setting [21].

Many secret-sharing-based secure computations are defined over a finite field, e.g., [6,4,13,7,11]. The choice of the underlying field greatly affects the efficiency of those protocols since field elements and field operations are the units of any processing, and the size of the field affects the size and efficiency of field elements and field operations, respectively. In other words, an unnecessarily large field incurs a large cost of storage, computation, and communication among parties. From this, a natural question arises: how small can the field size be?

Intuitively speaking, we can consider two lower bounds regarding the field size. The first is the range of values used in the computation. The field size should be large enough to contain any value that appears in all steps in the computation. For example, if one wants to store values that are less than 10, and compute the sum of 100 values, the field size should be larger than $10 \times 100 \approx 2^{10}$ to accommodate this computation, while if one wants to store binary values and compute a boolean circuit, a binary field is sufficient.

Another bound is derived from statistical errors which are typically a part of the computation, and which in turn provides an upper bound for the advantage of an adversary. These errors are typically dependent on the size of the field used in the computation. For example, consider a protocol for checking equality of secret shared values. Specifically, let $\mathbb{K}$ be a field, and let $[s]$ and $[s']$ be shares of $s, s' \in \mathbb{K}$. There is a straightforward way for the parties holding $[s]$ and $[s']$ to verify that $s = s'$ without revealing s and s' themselves: generate a random share $[r]$, securely compute $[r(s - s')]$, reconstruct this value, and verify whether the reconstructed value $r(s - s')$ is 0 or not. If $s \neq s'$, $r(s - s')$ will be a random value different from 0, except probability $1/|\mathbb{K}|$, where $|\mathbb{K}|$ denotes the size of $\mathbb{K}$. Therefore, if one wants to ensure that a statistical error probability is less than $2^{-\kappa}$, the field size must be larger than 2^{κ}.

The field size should be larger than these two lower bounds even if there is a gap between those. For example, if the parties securely compute statistics with a possible range from 0 to $1,000$ ($\approx 2^{10}$) with statistical error 2^{-40}, a field size larger than 2^{40} must be used (this comes from $\max(2^{10}, 2^{40})$).

Our Contribution. We propose a method to dynamically change the fields used in secure computation to improve efficiency. Note that a large field (e.g. chosen to lower the statistical error of a protocol) is not necessarily required in all stages of a secure computation. Often, a significant part of the computation can be done in a smaller field which is just sufficiently large to accommodate the values appearing in the computation, and only when verifying correctness, a much larger field is required to reduce the statistical error inherent in the used protocol, e.g. like the equality check described above.

Therefore, if we can dynamically change the underlying field, we can improve the efficiency of secure computation by using a field of an appropriate size for each stage of the computation. Note that for this approach to work, it is required that the parties can change the underlying field while a secret is shared over the field. In this paper, we propose a method that achieves this, which furthermore does not require the parties holding the shared secret to communicate. Hence, this allows the parties by default to use a small field over which computation is efficient, and only switch to a large (extended) field at the time of verification to achieve the desired statistical error.

Let us briefly recall standard construction of field extension. Let $\mathbb{K}$ be a base field and let $F \in \mathbb{K}[X]$ be an irreducible polynomial of degree $m - 1$. Then $\widehat{\mathbb{K}} := \mathbb{K}[X]/F$ is a field extension of $\mathbb{K}$ of size $|\mathbb{K}|^m$. An m-tuple of elements in $\mathbb{K}$, $(s_1, \ldots, s_m)$, can be regarded as a vector representation of a single element $\widehat{s} \in \widehat{\mathbb{K}}$ defined as $\widehat{s} = s_1 + s_2X + \cdots + s_mX^{m-1}$. Note that a single element $s_1 \in \mathbb{K}$ can also be regarded as an element in $\widehat{\mathbb{K}}$ by setting $s_i = 0$ for $2 \leq i \leq m$.

We show that this kind of extension allows shares from a secret sharing scheme to be mapped into the extended field, as long as we use a t-out-of-n linear secret sharing scheme. Let $[s]$ (that is not necessarily in $\mathbb{K}$) be a share of $s \in \mathbb{K}$ and $[\![s']\!]$ be a share of $s' \in \widehat{\mathbb{K}}$. We show that, if the parties have an m-tuple of shares $[s_1], \ldots, [s_m]$, the parties can regard them as a single share, $[\![s']\!]$, where $s' := s_1 + s_2X + \cdots + s_mX^{m-1}$. Similar to the above, this also implies that a single share $[s]$ can be regarded as a share of $[\![\widehat{s}]\!]$, where $\widehat{s} := s + 0X + \cdots + 0X^{m-1}$.

This technique is simple but useful for improving the efficiency of secure computation. Let us revisit the example of equality checking highlighted above. Assume that the parties have computed $[s]$ and $[s']$, where $s, s' \in \mathbb{K}$ and that to make the computation efficient $\mathbb{K}$ was chosen to be a small field, e.g., GF(2). Let $\widehat{\mathbb{K}}$ be the extended field of $\mathbb{K}$ with size larger than 2^{κ}. To check that $s = s'$, the parties extend $[s]$ and $[s']$ into $[\![\widehat{s}]\!]$ and $[\![\widehat{s'}]\!]$ using our technique, and generate a sharing $[\![r]\!]$ of randomness $r \in \widehat{\mathbb{K}}$. The parties then securely compute $[\![r(s - s')]\!]$, reconstruct the resulting value, and check whether this is 0 or not. Since the revealed value belongs to $\widehat{\mathbb{K}}$, the statistical error of the comparison, and thereby the advantage of an adversary, is bounded by $2^{-\kappa}$. Besides this, the parties can batch multiple equality checks by "packing" multiple secrets in $\mathbb{K}$ into a single secret in $\widehat{\mathbb{K}}$. If m secrets in a field are packed into a single element in an m-degree extended field, there is no extra cost with respect to communication compared to parallel executions of equality checks in $\mathbb{K}$. Similar scenarios appear in password-based threshold authentication [23] and batch consistency check [15,25], and we can apply this technique to the protocols for these.

As the main application of our technique, we show how to improve a recent protocol proposed by Chida et al. [11] which achieves fast large-scale honest-majority secure computation for active adversaries. Although Chida et al. proposed a protocol suitable for small fields, the protocol cannot be applied to a binary field, since the bound on the advantage of the adversary is given by $(3/|\mathbb{K}|)^{\delta}$, which is not meaningful for $|\mathbb{K}| = 2$. This, for example, prevents the use of XOR-free computation of a boolean circuit.[3] Informally, their protocol generates δ shared random values, computes δ "randomized" outputs in addition to the ordinary one, and verifies correctness of the computation by checking if the randomized outputs correspond to the ordinary outputs when the latter is randomized using the same randomnesses. Here, the shared random values and the randomized outputs are used for verification only. Hence, we can apply our technique to this protocol as follows. The (shared) random values and randomized outputs in $\mathbb{K}$ are replaced by a single random value and the randomized output in $\widehat{\mathbb{K}}$, and then the ordinary output in $\mathbb{K}$ is extended to $\widehat{\mathbb{K}}$ at the time of verification. The bound on the adversarial advantage in this modified protocol is $3/|\widehat{\mathbb{K}}|$. Therefore, we can choose a binary field as $\mathbb{K}$ (and an extension field $\widehat{\mathbb{K}}$ of appropriate size) in the protocol.

Related Work. There are several techniques to achieve active security even if the field size is small. Beaver [3] showed that one can securely compute a multiplication gate in the presence of active adversaries for any field/ring by sacrificing a multiplication triple. The SPDZ protocol [14] and subsequent studies generate the triples using cryptographic primitives, such as somewhat homomorphic encryption and oblivious transfer. Furukawa et al. [15] and subsequent works [1,25] used another approach to generate the triples using cut-and-choose technique in honest majority. Genkin et al. [17] introduced an algebraic manipulation detection (AMD) circuit that implies actively secure computation. Although their construction relies on a tamper-proof primitive if one uses a binary field, the later result [18] obtained a binary AMD circuit from a boolean one with polylog overhead with $2^{-\kappa}$ statistical error.

Cascudo et al. [10] employed different fields to offer a trade-off with the field size of the circuit. Their technique makes use of an encode between $\mathrm{GF}(2^k)$ and $(\mathrm{GF}(2))^m$ while maintaining the structure of multiplication of those ring and field in some sense. Since the motivation is different, our technique does not maintain the structure of multiplication, while our technique is space-efficient: we can embed several secrets in $\mathbb{K}$ into $\widehat{\mathbb{K}}$ without redundancy.

Paper Organization. The rest of the paper is organized as follows. In Section 2, we will recall linear secret sharing. In Section 3, we introduce our main technique of field extension in a secret-sharing form. In Section 4, we show several applications of our technique to secure computation based on threshold linear secret sharing: consistency check, equality check of multiple shares, and finally and as the main technical result, an efficient secure computation protocol for arithmetic circuits.

2 Linear Secret Sharing

In this section we give the definition of linear secret sharing [5]. Here, we consider general linear secret sharing with respect to an access structure.

Definition 2.1 (Secret Sharing). *Let $\mathbb{S}$ be a finite domain of secrets. Also, let $\mathcal{A}$ be an access structure of parties $P_1, \ldots, P_n$. A secret sharing scheme $\Pi = (\mathsf{Share}, \mathsf{Rec})$ realizing $\mathcal{A}$ satisfies the following two requirements:*

Reconstruction Requirement. *Let $\mathbb{S}_i$ be a finite domain of the party P_i's shares. For any set $G \in \mathcal{A}$ where $G = \{i_1, \ldots, i_{|G|}\}$, there exists a reconstruction function $\mathsf{Rec}_G : \mathbb{S}_{i_1} \times \cdots \times \mathbb{S}_{i_{|G|}} \to \mathbb{S}$ such that for any secret $s \in \mathbb{S}$, it holds that $\mathsf{Rec}_G([s]_{i_1}, \ldots, [s]_{i_{|G|}}) = s$ where $\mathsf{Share}(s) \to \langle [s]_1, \ldots, [s]_n \rangle$.*

[3] Precisely, secure computation in $\mathrm{GF}(2^m)$ is XOR-free but redundant for a boolean circuit.

Security Requirement. *For any set $B \notin \mathcal{A}$, any two secrets $\alpha, \beta \in \mathbb{S}$, and any elements $\mathsf{val}_i \in \mathbb{S}_i$ $(1 \le i \le n)$, it holds that*

$$\Pr[\ \bigwedge_{P_i \in B} \{\ [\alpha]_i = \mathsf{val}_i\ \}\ \ ] = \Pr[\ \bigwedge_{P_i \in B} \{\ [\beta]_i = \mathsf{val}_i\ \}\ \ ]$$

where the probabilities are taken over the randomness of the sharing algorithm.

Definition 2.2 (Linear Secret Sharing). *Let $\mathbb{K}$ be a finite field and Π a secret sharing scheme with a domain of secrets $\mathbb{S} \subseteq \mathbb{K}$ realizing an access structure $\mathcal{A}$ of parties $P_1, \ldots, P_n$. We say that Π is a linear secret sharing scheme over $\mathbb{K}$ if the following holds:*

1. *A share of each party consists of a vector over $\mathbb{K}$. More precisely for any index i, there exists a constant d_i such that the party P_i's share is taken from $\mathbb{K}^{d_i}$. We denote by $[s]_{ij}$ the j-th coordinate of the party P_i's share of a secret $s \in \mathbb{S}$.*
2. *For any set in $\mathcal{A}$, i.e., authorized set, its reconstruction function is linear. More precisely, for any set $G \in \mathcal{A}$, there exist constants $\{\alpha_{ij}\}_{P_i \in G, 1 \le j \le d_i}$ such that for any secret $s \in \mathbb{S}$, it holds that*

   $$s = \sum_{P_i \in G} \sum_{1 \le j \le d_i} \alpha_{ij} \cdot [s]_{ij}$$

   where the addition and multiplication are over the field $\mathbb{K}$.

If all shares consist of only one element in the field $\mathbb{K}$, Definition 2.2 implies that for any set $G \in \mathcal{A}$, there exist constants $\{\alpha_i\}_{P_i \in G}$ such that for any secret $s \in \mathbb{S}$, it holds that $s = \sum_{P_i \in G} \alpha_i \cdot [s]_i$.

3 Field Extension in Secret-Shared Form

In this section we propose a simple but highly useful method to extend a sharing of a secret over a field to a sharing of the same secret over an extended field, without requiring any communication between the parties which the secret is shared. This is the main mechanism we will exploit in our hybrid approach to protocol design, in which evaluation will be done over a smaller field, but verification is done over a large field to ensure a low statistical error, which in turn bounds the advantage of an adversary.

Let Π be a linear secret sharing scheme with a domain of secrets $\mathbb{S} \subseteq \mathbb{K}$ realizing an access structure $\mathcal{A}$ of parties $P_1, \ldots, P_n$. In the following, we consider a scenario in which these parties will be sharing m secrets $s_1, \ldots, s_m$.

Let $\widehat{\mathbb{K}} = \mathbb{K}[X]/F$ be the extended field of $\mathbb{K}$ where $F \in \mathbb{K}[X]$ is an irreducible polynomial of degree m. Let f be the bijective function of natural extension i.e. $f : \mathbb{K}^m \to \widehat{\mathbb{K}}$ and $f(a_1, \ldots, a_m) = a_1 + a_2 X + \cdots + a_m X^{m-1}$.

The following theorem shows that if secrets in $\mathbb{K}$ are shared via a linear secret sharing scheme as $[s_1]_{ij}, \ldots, [s_m]_{ij}$ with coefficients $\{\alpha_{ij}\}_{P_i \in G, 1 \le j \le d_i}$ for some $G \in \mathcal{A}$, a "packed" share $f\big([s_1]_{ij}, \ldots, [s_m]_{ij}\big)$ is in fact a share of $s_1 + s_2 X + \cdots + s_m X^m \in \widehat{\mathbb{K}}$ with coefficients $\{f(\alpha_{ij}, 0, \ldots, 0)\}_{P_i \in G, 1 \le j \le d_i}$. In other words, multiple shares can be embedded in the extended field $\widehat{\mathbb{K}}$ (which we will refer to as packing), and jointly reconstructed over $\widehat{\mathbb{K}}$. Since a party can locally compute $f\big([s_1]_{ij}, \ldots, [s_m]_{ij}\big)$, the parties can obtain a share of $s_1 + s_2 X + \cdots + s_m X^m \in \widehat{\mathbb{K}}$ from shares of $s_1, \ldots, s_m \in \mathbb{K}$ without communicating. The theorem also implies that the parties can obtain a share of $s_1 + \cdots + s_\ell X^\ell + 0 X^{\ell+1} + \cdots 0 X^m \in \widehat{\mathbb{K}}$ from shares of ℓ $(< m)$ secrets by setting $[s_k]_{ij} = 0$ for $\ell < k \le m$.

Theorem 3.1. *Let $[s_k]_{ij}$ be the j-th coordinate of P_i's share of a secret $s_k \in S$. Then for any set $G \in \mathcal{A}$, it holds that*

$$f^{-1}\Big(\sum_{P_i \in G} \sum_{1 \le j \le d_i} f(\alpha_{ij}, 0, \ldots, 0) \cdot f\big([s_1]_{ij}, \ldots, [s_m]_{ij}\big)\Big) = (s_1, \ldots, s_m)$$

where $\{\alpha_{ij}\}_{P_i \in G, 1 \le j \le d_i}$ are the constants defined in Definition 2.2.

[Proof] We have that

$$\begin{aligned}
&\sum_{P_i \in G} \sum_{1 \le j \le d_i} f(\alpha_{ij}, 0, \ldots, 0) \cdot f\big([s_1]_{ij}, \ldots, [s_m]_{ij}\big) \\
&\quad = \sum_{P_i \in G} \sum_{1 \le j \le d_i} \alpha_{ij} \cdot \Big([s_1]_{ij} + [s_2]_{ij} \cdot X + \cdots + [s_m]_{ij} \cdot X^{m-1}\Big) \\
&\quad = \sum_{P_i \in G} \sum_{1 \le j \le d_i} \alpha_{ij} \cdot [s_1]_{ij} + \Big(\sum_{P_i \in G} \sum_{1 \le j \le d_i} \alpha_{ij} \cdot [s_2]_{ij}\Big) X \\
&\qquad\qquad + \cdots + \Big(\sum_{P_i \in G} \sum_{1 \le j \le d_i} \alpha_{ij} \cdot [s_m]_{ij}\Big) X^{m-1} \\
&\quad = s_1 + s_2 \cdot X + \cdots + s_m \cdot X^{m-1}.
\end{aligned}$$

Thus, we see that $f^{-1}\Big(\sum_{P_i \in G} \sum_{1 \le j \le d_i} f(\alpha_{ij}, 0, \ldots, 0) \cdot f\big([s_1]_{ij}, \ldots, [s_m]_{ij}\big)\Big) = f^{-1}(s_1 + s_2 \cdot X + \cdots + s_m \cdot X^{m-1}) = (s_1, \ldots, s_m)$. ■

Induced Secret Sharing Scheme. The above theorem not only shows that shares from a secret sharing scheme over $\mathbb{K}$ can be embedded and reconstructed in the extension field $\widehat{\mathbb{K}}$, but in fact let us define an "induced" secret sharing scheme over $\widehat{\mathbb{K}}$ based on the secret sharing scheme over $\mathbb{K}$. More specifically, let $\Pi = (\mathsf{Share}, \mathsf{Rec})$ be a linear secret sharing scheme with a domain of secrets $S \subseteq \mathbb{K}$ realizing an access structure $\mathcal{A}$ of parties $P_1, \ldots, P_n$. We consider the induced scheme $\widehat{\Pi} = (\widehat{\mathsf{Share}}, \widehat{\mathsf{Rec}})$ with a domain of secrets $\widehat{S} \subseteq \widehat{\mathbb{K}} = \mathbb{K}[X]/F$ defined as follows.

$\widehat{\mathsf{Share}}(s)$:

1. Compute $(s_1, \ldots, s_m) \leftarrow f^{-1}(s)$
2. For $k \in [1, m]$: compute $\mathsf{Share}(s_k) \to \{[s_k]_{1j}\}_{1 \le j \le d_1}, \ldots, \{[s_k]_{nj}\}_{1 \le j \le d_n}\rangle$
3. For $i \in [1, n]$ and $j \in [1, d_i]$: set $[\![s]\!]_{ij} \leftarrow f([s_1]_{ij}, \ldots, [s_m]_{ij})$
4. Output $\langle\{[\![s]\!]_{1j}\}_{1 \le j \le d_1}, \ldots, \{[\![s]\!]_{nj}\}_{1 \le j \le d_n}\rangle$

$\widehat{\mathsf{Rec}}_G(\{[\![s]\!]_{ij}\}_{P_i \in G, j \in [1, d_i]})$:

1. For $P_i \in G$ and $j \in [1, d_i]$: compute $\widehat{\alpha_{ij}} \leftarrow f(\alpha_{ij}, 0, \ldots, 0)$.
2. Output $s \leftarrow \sum_{P_i \in G, j \in [1, d_i]} \widehat{\alpha_{ij}} \cdot [\![s]\!]_{ij}$.

The linearity of the above secret sharing scheme follows directly from Theorem 3.1, and security likewise follows in a straightforward manner. We write this as the following corollary.

Corollary 3.2. *Assume that Π is a linear secret sharing scheme. Then the induced secret sharing scheme $\widehat{\Pi}$ is a linear secret sharing scheme.*

The ability of the parties to locally evaluate the embedding function f, means that the parties can locally construct a sharing $[\![\widehat{s}]\!]$ of the induced scheme $\widehat{\Pi}$ from sharings $[s_1], \ldots, [s_m]$ of Π, where $\widehat{s} = s_1 + s_2 X + \cdots + s_m X^{m-1} \in \widehat{\mathbb{K}}$ and $s_1, \ldots, s_m \in \mathbb{K}$.

Throughout the paper, we will adopt the notation used above. Specifically, for a secret sharing scheme $\Pi = (\mathsf{Share}, \mathsf{Rec})$ over $\mathbb{K}$, which we will also refer to as the *base* scheme, $\widehat{\Pi} = (\widehat{\mathsf{Share}}, \widehat{\mathsf{Rec}})$ denotes the *induced* secret sharing scheme defined above over the field extension $\widehat{\mathbb{K}} = \mathbb{K}[X]/F$. For values $s \in \mathbb{K}$ and $v \in \widehat{\mathbb{K}}$, we will use $[s]$ and $[\![v]\!]$ to denote sharings of the base and the induced secret sharing scheme, respectively. We will sometimes abuse this notation, and for a value $s \in \mathbb{K}$ use $[\![s]\!]$ to denote $[\![f(s, 0, \ldots, 0)]\!]$, and will also refer to this as an induced sharing.

4 Applications to Secure Computation

In this section, we show several applications for actively secure computation with abort and an honest majority. As preliminaries to these, in Section 4.1, we first give basic definitions, including threshold secret sharing and several protocols that are used as building blocks. Then, we present applications of our field extension technique to consistency check of shares in Section 4.2, equality check of multiple shares in Section 4.3, and computation of arithmetic circuits in Section 4.4.

4.1 Preliminaries

Threshold Linear Secret Sharing. A t-out-of-n secret sharing scheme [5] enables n parties to share a secret $v \in \mathbb{K}$ so that no subset of t parties can learn any information about it, while any subset of $t+1$ parties can reconstruct it. In addition to being a linear secret sharing scheme, we require that the secret sharing scheme used in our protocol supports the following procedures:

- $\mathsf{Share}(v)$: We consider *non-interactive* secret sharing where there exists a probabilistic dealer D that receives a value v (and some randomness) and outputs shares $[v]_1, \ldots, [v]_n$. We denote the sharing of a value v by $[v]$. We use the notation $[v]_J$ to denote the shares held by a subset of parties $J \subset \{P_1, \ldots, P_n\}$. If the dealer is corrupted, then the shares received by the parties may not be correct. Nevertheless, we abuse notation and say that the parties hold shares $[v]$ even if these are not correct. We will define correctness of a sharing formally below.
- $\mathsf{Share}(v, [v]_J)$: This non-interactive procedure is similar to the previous one, except that here the shares of a subset J of parties with $|J| \leq t$ are fixed in advance. We assume that there exists a probabilistic algorithm $\widetilde{D}$ that receives a value v and some values $[v]_J = \{\widetilde{[v]}_i\}_{P_i \in J}$ (and some randomness) and outputs shares $[v]_1, \ldots, [v]_n$ where $[v]_i = \widetilde{[v]}_i$ holds for every $P_i \in J$. We also assume that if $|J| = t$, then $[v]_J$ together with v *fully determine* all shares. This also means that any $t+1$ shares fully determine all shares. (This follows since with $t+1$ shares one can always obtain v. However, for the secret sharing schemes we use, this holds directly as well.)
- $\mathsf{Reconstruct}([v], i)$: Given a sharing of a value v and an index i held by the parties, this interactive protocol guarantees that if $[v]$ is not correct (see formal definition below), then P_i will output $\perp$ and abort. Otherwise, if $[v]$ is correct, then P_i will either output v or abort.
- $\mathsf{Open}([v])$: Given a sharing of a value v held by the parties, this procedure guarantees that at the end of the execution, if $[v]$ is not correct, then *all* the honest parties will abort. Otherwise, if $[v]$ is correct, then each party will either output v or abort. Clearly, Open can be run by any subset of $t+1$ or more parties. We require that if any subset J of $t+1$ honest parties output a value v, then any superset of J will output either v or $\perp$ (but no other value).
- *Local Operations*: Given correct sharings $[u]$ and $[v]$, and a scalar $\alpha \in \mathbb{K}$, the parties can generate correct sharings of $[u+v]$, $[\alpha \cdot v]$ and $[v+\alpha]$ using local operations only (i.e., without any interaction). We denote these local operations by $[u]+[v], \alpha \cdot [v]$, and $[v]+\alpha$, respectively.

Standard secret sharing schemes like the Shamir scheme [27] and the replicated secret sharing scheme [22,12] support all of these procedures (with their required properties). Furthermore, if a base secret sharing scheme supports the above procedures, then the induced secret sharing scheme over a field extension likewise supports the above procedures. This easily follows from the one-to-one correspondence between a set of m shares $[s_1], \ldots, [s_m]$ of the base scheme and a share $[\![\widehat{s}]\!]$ of the induced scheme, where $\widehat{s} = s_1 + s_2 X + \cdots + s_m X^{m-1}$. Specifically, the above procedures for the induced scheme can be implemented by simply mapping the input shares to shares of the base scheme using f^{-1}, and running the corresponding procedure of the base scheme.

The following corollary regarding the security of an induced secret sharing scheme is a simple extension of Corollary 3.2 and follows from the one-to-one correspondence between a set of m shares in the base scheme and a share in the induced scheme.

Corollary 4.1. *Let Π be a secure threshold linear secret sharing scheme. Then the induced scheme $\widehat{\Pi}$ is a secure threshold linear secret sharing scheme.*

In the following, we set the threshold for the secret sharing scheme to be $\lfloor (n-1)/2 \rfloor$, and we denote by t the number of corrupted parties. Since we assume an honest majority, it holds that $t < n/2$, and so the corrupted parties can learn nothing about a shared secret.

We now define correctness for secret sharing. Let J be a subset of $t+1$ honest parties, and denote by $\mathsf{val}([v])_J$ the value obtained by these parties after running the Open procedure where no corrupted parties or additional honest parties participate. We note that $\mathsf{val}([v])_J$ may equal $\perp$ if the shares held by the honest parties are not valid. Informally, a secret sharing is correct if every subset of $t+1$ honest parties reconstruct the same value (which is not $\perp$). Formally:

Definition 4.2. *Let $H \subseteq \{P_1, \ldots, P_n\}$ denote the set of honest parties. A sharing $[v]$ is* correct *if there exists a value $\widetilde{v} \in \mathbb{K}$ ($\widetilde{v} \neq \perp$) such that for every $J \subseteq H$ with $|J| = t+1$ it holds that $\mathsf{val}([v])_J = \widetilde{v}$.*

If a sharing $[v]$ is not correct, then either there exists a subset J of $t{+}1$ honest users such that $\mathsf{val}([v])_J = \perp$, or there exists two subsets J_1 and J_2 such that $\mathsf{val}([v])_{J_1} = v_1$ and $\mathsf{val}([v])_{J_2} = v_2$, where $v_1, v_2 \in \mathbb{K}$ and $v_1 \neq v_2$. We will refer to the former as an $\mathsf{invalid}$ sharing, and the latter as a $\mathsf{value\text{-}inconsistent}$ sharing. Note that a correct sharing in an induced secret sharing scheme corresponds to a set of m correct shares of the base scheme (and conversely, if a single sharing in the base scheme is incorrect, the sharing in the induced scheme will be incorrect).

Definition of Security for Secure Computation. We use the standard definition of security based on the ideal/real model paradigm [9,19], with security formalized for non-unanimous abort. This means that the adversary first receives the output, and then determines for each honest party whether they will receive abort or receive their correct output. Also, we prove the security of our protocols in a hybrid model, where parties run a protocol with real messages and also have access to a trusted party computing a subfunctionality for them. The modular sequential composition of [8] states that one can replace the trusted party computing the subfunctionality with a real secure protocol computing the subfunctionality. When the subfunctionality is g, we say that the protocol works in the *g-hybrid model*. The formal definitions of security appears in Appendix A.

Definitions for Ideal Functionalities. Here, we recall the definitions of the ideal functionalities used in the paper, which are based on the ones used in [11]. These functionalities are associated with a threshold linear secret sharing scheme. Since in this paper we will utilize functionalities for both a secret sharing scheme for a base field $\mathbb{K}$ and those of the induced scheme for an extension field $\widehat{\mathbb{K}}$, we will use the style like $\mathcal{F}_{\mathrm{x}}$ for the former, and the style like $\widehat{\mathcal{F}}_{\mathrm{x}}$ for the latter. In the following, we only describe the functionalities for the base field $\mathbb{K}$; Those for the extension field $\widehat{\mathbb{K}}$ are defined in exactly the same way, with the correspondences that the sharing algorithm is of the induced scheme, and every value is of $\widehat{\mathbb{K}}$. We note that the protocols realizing these functionalities can be efficiently instantiated using standard secret sharing schemes [27,12,2]. (These protocols treat the underlying field and secret sharing scheme in a black-box manner, and hence can be naturally used for realizing the functionalities for the induced scheme.)

- $\mathcal{F}_{\mathrm{coin}}$ – *Generating Random Coins:* When invoked, this functionality picks an element $r \in \mathbb{K}$ uniformly at random and sends it to all parties.
- $\mathcal{F}_{\mathrm{rand}}$ – *Generating Random Shares:* This functionality generates a sharing of a random value in $\mathbb{K}$ unknown to the parties. The formal description is given in Functionality 4.3.
- $\mathcal{F}_{\mathrm{input}}$ – *Secure Sharing of Inputs:* This functionality captures a secure sharing of the parties' inputs. The formal description is given in Functionality 4.4.
- $\mathcal{F}_{\mathrm{checkZero}}$ – *Checking Equality to 0:* This functionality allows callers to check whether a given sharing is a sharing of 0 without revealing any further information on the shared value. The formal description is given in Functionality 4.5.
- $\mathcal{F}_{\mathrm{mult}}$ – *Secure Multiplication up to Additive Attacks [17,16]:* This functionality captures a secure computation of a multiplication gate in an arithmetic circuit, but allows an adversary to mount the so-called *additive attacks*. Specifically, this functionality receives input sharings $[x]$ and $[y]$ from the honest parties

FUNCTIONALITY 4.3 ($\mathcal{F}_{\text{rand}}$ – Generating Random Shares)

Upon receiving $\{\alpha_i\}_{P_i \in \mathcal{C}}$ from the ideal adversary $\mathcal{S}$, $\mathcal{F}_{\text{rand}}$ chooses a random $r \in \mathbb{K}$, sets $[r]_{\mathcal{C}} = \{\alpha_i\}_{P_i \in \mathcal{C}}$, and runs $[r] = ([r]_1, \ldots, [r]_n) \leftarrow \mathsf{Share}(r, [r]_{\mathcal{C}})$. Then, $\mathcal{F}_{\text{rand}}$ hands each honest party P_i (for $i \in H$) its share $[r]_i$.

FUNCTIONALITY 4.4 ($\mathcal{F}_{\text{input}}$- Sharing of Inputs)

1. $\mathcal{F}_{\text{input}}$ receives inputs $v_1, \ldots, v_M \in \mathbb{K}$ from the parties. For each $k \in \{1, \ldots, M\}$, $\mathcal{F}_{\text{input}}$ also receives from the ideal adversary $\mathcal{S}$ the corrupted parties' shares $[v_k]_{\mathcal{C}}$ for the k-th input.
2. For each $k \in \{1, \ldots, M\}$, $\mathcal{F}_{\text{input}}$ runs $[v_k] = ([v_k]_1, \ldots, [v_k]_n) \leftarrow \mathsf{Share}(v_k, [v_k]_{\mathcal{C}})$.
3. For each $i \in \{1, \ldots, n\}$, $\mathcal{F}_{\text{input}}$ sends P_i the shares $([v_1]_i, \ldots, [v_M]_i)$.

FUNCTIONALITY 4.5 ($\mathcal{F}_{\text{checkZero}}$ – Checking Equality to 0)

$\mathcal{F}_{\text{checkZero}}$ receives $[v]_H$ from the honest parties and uses it to compute v. Then:

- If $v = 0$, then $\mathcal{F}_{\text{checkZero}}$ sends 0 to the ideal adversary $\mathcal{S}$. Then, if $\mathcal{S}$ sends *reject* (resp. *accept*), then $\mathcal{F}_{\text{checkZero}}$ sends *reject* (resp. *accept*) to the honest parties.
- If $v \neq 0$, then $\mathcal{F}_{\text{checkZero}}$ proceeds as follows:
  - With probability $1/|\mathbb{K}|$, it sends *accept* to the honest parties and $\mathcal{S}$.
  - With probability $1 - 1/|\mathbb{K}|$, it sends *reject* to the honest parties and $\mathcal{S}$.

FUNCTIONALITY 4.6 ($\mathcal{F}_{\text{mult}}$ - Secure Multiplication up to Additive Attacks)

1. Upon receiving $[x]_H$ and $[y]_H$ from the honest parties where $x, y \in \mathbb{K}$, $\mathcal{F}_{\text{mult}}$ computes x, y and the corrupted parties' shares $[x]_{\mathcal{C}}$ and $[y]_{\mathcal{C}}$.
2. $\mathcal{F}_{\text{mult}}$ hands $[x]_{\mathcal{C}}$ and $[y]_{\mathcal{C}}$ to the ideal adversary $\mathcal{S}$.
3. Upon receiving d and $\{\alpha_i\}_{P_i \in \mathcal{C}}$ from $\mathcal{S}$, $\mathcal{F}_{\text{mult}}$ defines $z = x \cdot y + d$ and $[z]_{\mathcal{C}} = \{\alpha_i\}_{P_i \in \mathcal{C}}$. Then, $\mathcal{F}_{\text{mult}}$ runs $[z] = ([z]_1, \ldots, [z]_n) \leftarrow \mathsf{Share}(z, [z]_{\mathcal{C}})$.
4. $\mathcal{F}_{\text{mult}}$ hands each honest party P_i its share $[z]_i$.

FUNCTIONALITY 4.7 ($\mathcal{F}_{\text{product}}$ - Secure Sum-of-Products up to Additive Attacks)

1. Upon receiving $\{[x_\ell]_H\}_{\ell=1}^{L}$ and $\{[y_\ell]_H\}_{\ell=1}^{L}$ from the honest parties where $x_\ell, y_\ell \in \mathbb{K}$, $\mathcal{F}_{\text{product}}$ computes x_ℓ and y_ℓ and the corrupted parties' shares $[x_\ell]_{\mathcal{C}}$ and $[y_\ell]_{\mathcal{C}}$, for each $\ell \in \{1, \ldots, L\}$.
2. $\mathcal{F}_{\text{product}}$ hands $\{[x_\ell]_{\mathcal{C}}\}_{\ell=1}^{L}$ and $\{[y_\ell]_{\mathcal{C}}\}_{\ell=1}^{L}$ to the ideal adversary $\mathcal{S}$.
3. Upon receiving d and $\{\alpha_i\}_{P_i \in \mathcal{C}}$ from $\mathcal{S}$, $\mathcal{F}_{\text{product}}$ defines $z = \sum_{\ell=1}^{L} x_\ell \cdot y_\ell + d$ and $[z]_{\mathcal{C}} = \{\alpha_i\}_{P_i \in \mathcal{C}}$. Then, it runs $[z] = ([z]_1, \ldots, [z]_n) \leftarrow \mathsf{Share}(z, [z]_{\mathcal{C}})$.
4. $\mathcal{F}_{\text{product}}$ hands each honest party P_i its share $[z]_i$.

and an additive value d from the adversary, and outputs a sharing of $x \cdot y + d$. The formal description is given in Functionality 4.6.

– $\mathcal{F}_{\text{product}}$ – *Secure Sum of Products up to Additive Attacks:* This functionality captures a secure computation for the inner product of two vectors of input sharings. As with $\mathcal{F}_{\text{mult}}$, security up to additive attacks is considered. The formal description is given in Functionality 4.7.

4.2 Share Consistency Check

In this section, we present a protocol for checking the correctness of a collection of shares $[x_1], \ldots, [x_l]$. The protocol outputs *reject* if there is an invalid or incorrect share in $[x_1], \ldots, [x_l]$, and outputs *accept* otherwise. The protocol is based on Protocol 3.1 from [25], and works by choosing random coefficients from the extension field, using these to compute a linear combination of the shares embedded in the extension field, and finally opening the resulting sharing. To ensure no information regarding the original shares is revealed, a sharing of a random value of the extension field is added to the linear combination of shares. The description of the protocol is shown in Protocol 4.10. Note that, unlike [25], the coefficients for the linear combination are chosen from the full extension field, which allows an analysis with a better probability bound; while the original protocol from [25] will fail with probability $\frac{1}{|\mathbb{K}|-1}$, our protocol fails with probability $\frac{1}{|\hat{\mathbb{K}}|}$. Hence, our protocol will, in addition to allowing the failure probability to be freely adjusted via the size of the extension field, also remain meaningful for binary fields, for which the original protocol cannot be used.

The protocol relies on the base secret sharing scheme to be *robustly-linear*, which is defined as follows.

Definition 4.8. *A secret sharing scheme is robustly-linear if for every pair of invalid shares $[u]$ and $[v]$, there exists a unique $\alpha \in \mathbb{K}$ such that $\alpha[u] + [v]$ is valid (when computed locally by the parties).*

Note that secret sharing schemes for which there are no invalid shares, like the Shamir secret sharing scheme, will trivially be robustly-linear.

The following lemma plays a central role in the analysis of Protocol 4.10.

Lemma 4.9. *Let $[u]$ be an incorrect sharing of a robustly-linear secret sharing scheme over $\mathbb{K}$, and let $[\![v]\!]$ be any sharing of the induced secret sharing scheme over $\widehat{\mathbb{K}}$. Then, for a randomly chosen $\alpha \in \widehat{\mathbb{K}}$, the probability that $\alpha \cdot f([u], 0, \ldots, 0) + [\![v]\!]$ is a correct sharing, is at most $1/|\widehat{\mathbb{K}}|$.*

[**Proof**] The proof proceeds by considering the possible combinations of validity, invalidity, and value-inconsistency of $[u]$ and $[\![v]\!]$, and for each combination, show that only a single choice of $\alpha \in \widehat{\mathbb{K}}$ will make $[\![w]\!] = \alpha \cdot f([u], 0, \ldots, 0) + [\![v]\!]$ a valid sharing.

Firstly, recall that a value $w \in \widehat{\mathbb{K}}$ can be expressed as $w = w_1 + w_2X + \cdots + w_mX^{m-1}$, where $w_i \in \mathbb{K}$, and a sharing $[\![w]\!]$ in the induced sharing scheme over $\widehat{\mathbb{K}}$ corresponds to $[\![w]\!] = [w_1] + [w_2]X + \cdots + [w_m]X^{m-1}$, where $[w_i]$ are shares over $\mathbb{K}$. Note that for a sharing $[\![w]\!]$ to be valid, each sharing $[w_i]$ for $1 \le i \le m$ must be valid.

Now consider $[\![w]\!] = \alpha f([u], 0, \ldots, 0) + [\![v]\!]$ for a value $\alpha \in \widehat{\mathbb{K}}$, and let $\alpha_i \in \mathbb{K}$ for $1 \le i \le m$ be the values defining α. Then, it must hold that $[w_i] = \alpha_i[u] + [v_i]$. In the following, we will argue about the validity of $[w_i]$. We consider the following cases.

- $[v_i]$ is valid. In this case, only $\alpha_i = 0$ will make $[w_i]$ valid. To see this, assume for the purpose of a contradiction, that $[w_i]$ is valid and $\alpha_i \neq 0$. Then $[u] = \alpha_i^{-1}([w_i] - [v_i])$ will be valid due to the validity of local computations, which contradicts the assumption in the lemma that $[u]$ is incorrect.
- $[v_i]$ is value-inconsistent. That is, there exist sets J_1 and J_2 of $t+1$ users such that $\mathsf{val}([v_i])_{J_1} = v_i^{(1)}$, $\mathsf{val}([v_i])_{J_2} = v_i^{(2)}$, and $v_i^{(1)} \neq v_i^{(2)}$. There are two sub-cases to consider, $[u]$ being value-inconsistent or invalid (recall that the assumption in the lemma is that $[u]$ is incorrect).
  - $[u]$ is value-inconsistent. Let $\mathsf{val}([u])_{J_1} = u^{(1)}$ and $\mathsf{val}([u]))_{J_2} = u^{(2)}$. Note that $\mathsf{val}([w_i])_{J_1} = \alpha_i u^{(1)} + v_i^{(1)}$ and $\mathsf{val}([w_i])_{J_1} = \alpha_i u^{(1)} + v_i^{(1)}$ due to the correctness of local operations. Now, if $u^{(1)} = u^{(2)}$, it must hold that $\alpha_i u^{(1)} + v_i^{(1)} \neq \alpha_i u^{(2)} + v_i^{(2)}$, since $v_i^{(1)} \neq v_i^{(2)}$. Hence, $[w_i]$ is value-inconsistent. On the other hand, if $u^{(1)} \neq u^{(2)}$, only the unique value $\alpha_i = \frac{v_i^{(2)} - v_i^{(1)}}{u^{(1)} - u^{(2)}}$ will ensure that $\alpha_i u^{(1)} + v_i^{(1)} = \alpha_i u^{(2)} + v_i^{(2)}$, and thereby make $[w_i]$ valid.
  - $[u]$ is invalid. Firstly, observe that $\alpha_i = 0$ implies that $[w_i] = [v_i]$, and as $[v_i]$ is value-inconsistent, so will be $[w_i]$. Hence, in the following analysis assumes that $\alpha_i \neq 0$. Since $[u]$ is invalid, there is a set J' satisfying $\mathsf{val}([u])_{J'} = \perp$. For this J' we claim that $\mathsf{val}([w_i])_{J'} = \perp$. To see this assume that $\mathsf{val}([w_i])_{J'} \neq \perp$. Since $[v_i]$ is value-inconsistent, $\mathsf{val}([v_i])_{J'} \neq \perp$. Thus we have that $\mathsf{val}(\alpha_i^{-1}([w_i] - [v_i]))_{J'} = \mathsf{val}([u])_{J'} \neq \perp$, which contradicts the definition of J'.
- $[v_i]$ is invalid. There are again two sub-cases to consider.
  - $[u]$ is value-inconsistent. This case is symmetric to the case where $[v_i]$ is value-inconsistent and $[u]$ is invalid. A similar analysis to the above yields that at most a single choice of α_i will make $[w_i]$ valid.
  - $[u]$ is invalid. That is, both $[u]$ and $[v_i]$ are invalid. As the secret sharing scheme over $\mathbb{K}$ is assumed to be robustly-linear, there is only a single value α_i that will make $[w_i] = \alpha_i[u] + [v_i]$ valid.

As shown in the above analysis, all possible combinations of validity, value-inconsistency, and invalidity of $[u]$ and $[v_i]$ lead to at most a single possible value α_i that will make $[w_i]$ valid. Since α is picked uniformly at random from $\widehat{\mathbb{K}}$, the α_i values are independent and uniformly distributed in $\mathbb{K}$. Hence, the probability that $[\![w]\!]$ is valid, which requires each $[w_i]$ to be valid, is bounded by $(1/|\mathbb{K}|)^m = 1/|\widehat{\mathbb{K}}|$. ■

PROTOCOL 4.10 (Share Consistency Check)

Inputs: The parties hold l shares $[x_1], \ldots, [x_l]$.

Auxiliary Input: The parties hold the description of finite fields $\mathbb{K}$ and $\widehat{\mathbb{K}}$.

The protocol:

1. For all $i \in [l]$, the parties compute $[\![x_i]\!] = f([x_i], 0, \ldots, 0)$.
2. The parties call $\widehat{\mathcal{F}}_{\text{coin}}$ to obtain random elements $\alpha_1, \ldots, \alpha_l \in \widehat{\mathbb{K}}$.
3. The parties call $\widehat{\mathcal{F}}_{\text{rand}}$ to obtain a sharing $[\![r]\!]$ for a random element $r \in \widehat{\mathbb{K}}$.
4. The parties locally compute
$$[\![w]\!] = \alpha_1 \cdot [\![x_1]\!] + \ldots + \alpha_l \cdot [\![x_l]\!] + [\![r]\!]$$
5. The parties run $\mathsf{Open}([\![w]\!])$.
6. If any party aborts, the parties output *reject*. Otherwise, the parties output *accept*.

With the above lemma in place, establishing the following result is straightforward.

Theorem 4.11. *Assume the sharing scheme over $\mathbb{K}$ is robustly-linear. Then, in Protocol 4.10, if one of the input shares $[x_1], \ldots, [x_l]$ is not correct, the honest parties in the protocol will output accept with probability at most $1/|\widehat{\mathbb{K}}|$.*

[**Proof**] Assume that there is an index $i \in [l]$ such that $[x_i]$ is not correct, and note that $[\![v]\!]$ can be expressed as $[\![w]\!] = \alpha_i f([x_i], 0, \ldots, 0) + [\![v]\!]$, where $[\![v]\!] = \sum_{j \in [l] \setminus \{i\}} \alpha_j f([x_j], 0, \ldots, 0) + [\![r]\!]$. Then, applying Lemma 4.9 yields that, when $\alpha_i \in \widehat{\mathbb{K}}$ is picked uniformly at random, as done in the protocol, the probability that $[\![w]\!]$ is correct, is at most $1/|\widehat{\mathbb{K}}|$. As Open guarantees that the honest parties will output reject on input an incorrect share, the theorem follows. ■

Similar to [25], we will not define the ideal functionality and show full security of Protocol 4.10, as this leads to complications. For example, defining the ideal functionality would require knowing how to generate the inconsistent messages caused by inconsistent shares. Instead, the protocol will have to be simulated directly when showing security of a larger protocol using Protocol 4.10 as a sub-protocol.

4.3 Equality Check of Multiple Shares

Here, we show a simple application of our field extension technique to a protocol for checking that multiple shared secrets $[v_1], \ldots, [v_m]$ of the base field elements $v_1, \ldots, v_m \in \mathbb{K}$ are all equal to 0. This functionality, which we denote by $\mathcal{F}_{\mathsf{mcheckZero}}$, is specified in Functionality 4.12. Our protocol uses the ideal functionality $\widehat{\mathcal{F}}_{\text{checkZero}}$ (Functionality 4.5) in a straightforward way, and thus the definition of $\mathcal{F}_{\mathsf{mcheckZero}}$ incorporates an error probability from the false positive case, namely, even if some non-zero shared secret is contained in the inputs, the protocol outputs *accept* with probability at most $1/|\widehat{\mathbb{K}}|$, where $\widehat{\mathbb{K}}$ is the extension field. The formal description of our protocol appears in Protocol 4.13.

FUNCTIONALITY 4.12 ($\mathcal{F}_{\mathsf{mcheckZero}}$ — Batch-Checking Equality to 0)

$\mathcal{F}_{\mathsf{mcheckZero}}$ receives $[v_1]_H, \ldots, [v_m]_H$ from the honest parties and uses them to compute $v_1, \ldots, v_m$. Then,

1. If $v_1 = \cdots = v_m = 0$, then $\mathcal{F}_{\mathsf{mcheckZero}}$ sends 0 to the ideal adversary $\mathcal{S}$. Then, if $\mathcal{S}$ sends *reject* (resp., *accept*), then $\mathcal{F}_{\mathsf{mcheckZero}}$ sends *reject* (resp., *accept*) to the honest parties.
2. If $v_i \neq 0$ for some $i \in \{1, \ldots, m\}$, then $\mathcal{F}_{\mathsf{mcheckZero}}$ proceeds as follows:
   (a) With probability $1/|\widehat{\mathbb{K}}|$, it sends *accept* to the honest parties and $\mathcal{S}$.
   (b) With probability $1 - 1/|\widehat{\mathbb{K}}|$, it sends *reject* to the honest parties and $\mathcal{S}$.

PROTOCOL 4.13 (Batch-Checking Equality to 0)

Inputs: The parties hold a sharing $[v_1]$, ..., $[v_m]$.

The protocol:

1. The parties locally compute a "packed" share $[\![\widehat{v}]\!] = f([v_1], \ldots, [v_m])$.
2. The parties call $\widehat{\mathcal{F}}_{\text{checkZero}}$ on input $[\![\widehat{v}]\!]$, and output whatever $\widehat{\mathcal{F}}_{\text{checkZero}}$ outputs.

The security of Protocol 4.13 is guaranteed by the following theorem. (We omit the proof since it is straightforward.)

Theorem 4.14. *Protocol 4.13 securely computes $\mathcal{F}_{\mathsf{mcheckZero}}$ with abort in the $\widehat{\mathcal{F}}_{\text{checkZero}}$-hybrid model in the presence of active adversaries who control $t < n/2$ parties.*

Since the efficient protocol for checking equality to zero of a finite field $\mathbb{K}$ by Chida et al. [11, Protocol 3.7] in the $(\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{mult}})$-hybrid model uses the underlying secret sharing scheme and the finite field in a black-box manner, it can be used as one for checking equality to zero for an extension field $\widehat{\mathbb{K}}$ in the $(\widehat{\mathcal{F}}_{\text{rand}}, \widehat{\mathcal{F}}_{\text{mult}})$-hybrid model. Hence, by combining this protocol with Theorem 4.14, we also obtain an efficient protocol for checking equality to zero of multiple shared secrets in the base field $\mathbb{K}$ in the $(\widehat{\mathcal{F}}_{\text{rand}}, \widehat{\mathcal{F}}_{\text{mult}})$-hybrid model.

An obvious merit of our protocol is that it can be used even if the size of the base field $\mathbb{K}$ is small, i.e, $|\mathbb{K}| \leq 2^{\kappa}$ for an intended statistical error κ. On the contrary, Chida et al.'s original protocol [11, Protocol 3.7] cannot be used for small field elements. Another merit of our protocol is that by adjusting the size of the extension field $\widehat{\mathbb{K}}$, we can flexibly reduce the error probability (i.e. the false positive probability) that the protocol outputs *accept* even though some input shares contain a non-zero secret.

Application to Password-Based Authentication. We can apply our protocol to implement a password-based authentication protocol, such as [23]. Let us consider the following scenario. A password is stored among multiple backend servers in a linear secret sharing form. To log-in to the system the user splits his password into shares and sends each share to each server. The servers run Protocol 4.13 to determine whether the password sent from the user is correct (more precisely, the servers subtract the two shares, the one sent from the user and the one stored by themselves, and run Protocol 4.13 to determine whether the difference between two shares is zero).

We claim that the most space-efficient way to store the password is to store a password in a character-by-character manner. For example, if a password is encoded by the ASCII code (8-bit represents a single character), shares of a password is a sequence of $\mathrm{GF}(2^8)$-shares. By running Protocol 4.13, the servers combine $\mathrm{GF}(2^8)$-shares into a single induced share which contains the entire password, and check that the secret-shared bytes are all zeros. This approach has advantages over the following alternative choices regarding storage capacity. The first alternative is (1) to use a field sufficiently large both for storing the password and for providing statistical security, for example, a 320-bit field. This alternative is not efficient because we need to allocate 320 bits of storage for every password, which may include short, say, 8-byte passwords. Another alternative is (2) to use a field sufficiently large for statistical security but not necessarily large enough for storing the password, for example, a 40-bit field. In this case, the password will be stored by first dividing the password into a sequence of 40-bit blocks, and then share these among the servers in a block-by-block manner. This alternative is again not efficient, particularly in the case that the length of a password is not a multiple of the size of a block.

4.4 Secure Computation for Arithmetic Circuits

As mentioned earlier, the original highly efficient protocol for computing arithmetic circuits for a small finite field by Chida et al. [11, Protocol 5.3], in fact cannot be used for a field $\mathbb{K}$ with $|\mathbb{K}| \leq 3$ (e.g. computation for boolean circuits).

In this section, we show how to remove this restriction by using our field extension technique. Namely, we propose a variant of Chida et al.'s protocol that truly works for any finite field. The formal description of our protocol is given in Protocol 4.15. The simple idea employed in our protocol is to perform the computations for the randomized shares $[r \cdot x]$ and the equality check of the invariant done in the verification stage, over an extension field $\widehat{\mathbb{K}}$. In contrast, these operations are done over the base field $\mathbb{K}$ in Chida et al.'s original protocol. This allows us to perform the computation of the randomized shares using only a single element (of the extension field), while still achieving statistical security $3/|\widehat{\mathbb{K}}|$, which is a simplification compared to the protocol by Chida et al. Note that $3/|\widehat{\mathbb{K}}|$ can be chosen according to the desired statistical error by adjusting the degree m for the field extension.

PROTOCOL 4.15 (Computing Arithmetic Circuits over Any Finite $\mathbb{K}$)

Inputs: Each party P_i ($i \in \{1, \dots, n\}$) holds an input $x_i \in \mathbb{K}^\ell$.

Auxiliary Input: The parties hold the description of finite fields $\mathbb{K}$ and $\widehat{\mathbb{K}}$ with $3/|\widehat{\mathbb{K}}| \leq 2^{-\kappa}$, and an arithmetic circuit C over $\mathbb{K}$ that computes $\mathcal{F}$ on inputs of length $M = \ell \cdot n$. Let N be the number of multiplication gates in C.

The protocol:

1. *Secret sharing the inputs:* For each input v_j held by the party P_i, the party P_i sends v_j to $\mathcal{F}_{\text{input}}$. Each party P_i records its vector of shares $([v_1]_i, \dots, [v_M]_i)$ of all inputs, as received from $\mathcal{F}_{\text{input}}$. If the party received $\perp$ from $\mathcal{F}_{\text{input}}$, then it sends abort to the other parties and halts.
2. *Generate a randomizing share:* The parties call $\widehat{\mathcal{F}}_{\text{rand}}$ to receive a sharing $[\![\widehat{r}]\!]$.
3. *Randomization of inputs:* For each input wire sharing $[v_j]$ (where $j \in \{1, \dots, M\}$), the parties locally compute the induced share $[\![v_j]\!] = f([v_j], 0, \dots, 0)$. Then, the parties call $\widehat{\mathcal{F}}_{\text{mult}}$ on $[\![\widehat{r}]\!]$ and $[\![v_j]\!]$ to receive $[\![\widehat{r} \cdot v_j]\!]$.
4. *Circuit emulation:* Let $G_1, \dots, G_{|C|}$ be a predetermined topological ordering of the gates of the circuit C. For $j = 1, \dots, |C|$ the parties proceed as follows:
   – *If G_j is an addition gate:* Given pairs $([x], [\![\widehat{r} \cdot x]\!])$ and $([y], [\![\widehat{r} \cdot y]\!])$ on the *left* and *right* input wires respectively, each party locally computes $([x + y], [\![\widehat{r} \cdot (x + y)]\!])$.
   – *If G_j is a multiplication-by-a-constant gate:* Given a pair $([x], [\![\widehat{r} \cdot x]\!])$ on the input wire and a constant $a \in \mathbb{K}$, each party locally computes $([a \cdot x], [\![\widehat{r} \cdot (a \cdot x)]\!])$.
   – *If G_j is a multiplication gate:* Given pairs $([x], [\![\widehat{r} \cdot x]\!])$ and $([y], [\![\widehat{r} \cdot y]\!])$ on the *left* and *right* input wires respectively, the parties compute $([x \cdot y], [\![\widehat{r} \cdot x \cdot y]\!])$ as follows:
     (a) The parties call $\mathcal{F}_{\text{mult}}$ on $[x]$ and $[y]$ to receive $[x \cdot y]$.
     (b) The parties locally compute the induced share $[\![y]\!] = f([y], 0, \dots, 0)$.
     (c) The parties call $\widehat{\mathcal{F}}_{\text{mult}}$ on $[\![\widehat{r} \cdot x]\!]$ and $[\![y]\!]$ to receive $[\![\widehat{r} \cdot x \cdot y]\!]$.
5. *Verification stage:* Let $\{([z_k], [\![\widehat{r} \cdot z_k]\!])\}_{k=1}^N$ be the pairs on the output wires of the multiplication gates, and $\{([v_j], [\![\widehat{r} \cdot v_j]\!])\}_{j=1}^M$ be the pairs on the input wires of C.
   (a) For $k = 1, \dots, N$, the parties call $\widehat{\mathcal{F}}_{\text{rand}}$ to receive $[\![\widehat{\alpha}_k]\!]$.
   (b) For $j = 1, \dots, M$, the parties call $\widehat{\mathcal{F}}_{\text{rand}}$ to receive $[\![\widehat{\beta}_j]\!]$.
   (c) *Compute linear combinations:*
      i. The parties call $\widehat{\mathcal{F}}_{\text{product}}$ on vectors $([\![\widehat{\alpha}_1]\!], \dots, [\![\widehat{\alpha}_N]\!], [\![\widehat{\beta}_1]\!], \dots, [\![\widehat{\beta}_M]\!])$ and $([\![\widehat{r} \cdot z_1]\!], \dots, [\![\widehat{r} \cdot z_N]\!], [\![\widehat{r} \cdot \widehat{v}_1]\!], \dots, [\![\widehat{r} \cdot v_M]\!])$ to receive $[\![\widehat{u}]\!]$.
      ii. For each $k \in \{1, \dots, N\}$, the parties locally compute the induced share $[\![z_k]\!] = f([z_k], 0, \dots, 0)$ of the output wire of the k-th multiplication gate. Then, the parties call $\widehat{\mathcal{F}}_{\text{product}}$ on vectors $([\![\widehat{\alpha}_1]\!], \dots, [\![\widehat{\alpha}_N]\!], [\![\widehat{\beta}_1]\!], \dots, [\![\widehat{\beta}_M]\!])$ and $([\![z_1]\!], \dots, [\![z_N]\!], [\![v_1]\!], \dots, [\![v_M]\!])$ to receive $[\![\widehat{w}]\!]$.
      iii. The parties run $\mathsf{Open}([\![\widehat{r}]\!])$ to receive $\widehat{r}$.
      iv. Each party locally computes $[\![\widehat{T}]\!] = [\![\widehat{u}]\!] - \widehat{r} \cdot [\![\widehat{w}]\!]$.
      v. The parties call $\widehat{\mathcal{F}}_{\text{checkZero}}$ on $[\![\widehat{T}]\!]$. If $\widehat{\mathcal{F}}_{\text{checkZero}}$ outputs *reject*, the parties output $\perp$ and abort. Else, if it outputs *accept*, they proceed.
6. *Output reconstruction:* For each output wire of C, the parties run $\mathsf{Reconstruct}([v], i)$ where $[v]$ is the sharing on the output wire, and P_i is the party whose output is on the wire. If a party received $\perp$ in any of the Reconstruct procedures, it sends $\perp$ to the other parties, outputs $\perp$, and halts.

Output: If a party has not aborted, it outputs the values received on its output wires.

The following theorem formally guarantees the security of our protocol.

Theorem 4.16. *Let κ be a statistical security parameter such that $3/|\widehat{\mathbb{K}}| \leq 2^{-\kappa}$. Let $\mathcal{F}$ be an n-party functionality over $\mathbb{K}$. Then, Protocol 4.15 securely computes $\mathcal{F}$ with abort in the $(\mathcal{F}_{\text{input}}, \mathcal{F}_{\text{mult}}, \widehat{\mathcal{F}}_{\text{mult}}, \widehat{\mathcal{F}}_{\text{product}}, \widehat{\mathcal{F}}_{\text{rand}}, \widehat{\mathcal{F}}_{\text{checkZero}})$-hybrid model with statistical error $2^{-\kappa}$, in the presence of active adversaries who control $t < n/2$ parties.*

Before giving the proof of Theorem 4.16, we give the key lemma that states that in the real protocol execution, if an adversary $\mathcal{A}$ uses some non-zero value in its additive attacks in the invocations of $\mathcal{F}_{\text{mult}}$, $\widehat{\mathcal{F}}_{\text{mult}}$ and $\widehat{\mathcal{F}}_{\text{product}}$, the probability that the honest parties can detect it via the use of $\widehat{\mathcal{F}}_{\text{checkZero}}$ except with probability at most $2/|\widehat{\mathbb{K}}|$.

Lemma 4.17. *In the real execution of Protocol 4.15, if an adversary $\mathcal{A}$ sends some non-zero additive value in any of the calls to $\mathcal{F}_{\text{mult}}$, $\widehat{\mathcal{F}}_{\text{mult}}$, or $\widehat{\mathcal{F}}_{\text{product}}$, then the value $[\![\widehat{T}]\!]$ computed in Verification stage (Step 5) equals 0 with probability less than $2/|\widehat{\mathbb{K}}|$.*

The proof of the lemma proceeds closely to that of [11, Lemma 4.2]. Before going to the formal proof, we give its overview. Note that during an execution of the protocol, $\mathcal{A}$ may mount additive attacks when calling $\widehat{\mathcal{F}}_{\text{mult}}$ in Step 3, calling $\mathcal{F}_{\text{mult}}$ and $\widehat{\mathcal{F}}_{\text{mult}}$ at the computation of multiplication gates in Step 4, and calling $\widehat{\mathcal{F}}_{\text{product}}$ at (c) in Step 5. We classify the cases at which stage $\mathcal{A}$ uses a non-zero additive value in its additive attacks for the first time: (1) Step 3, (2) Step 4, or (3) Step 5. Then, for each case, we can show that the probability that $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ holds is at most $2/|\widehat{\mathbb{K}}|$. More specifically, we can show that the probability of $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ occurring in Case (1) is exactly $1/|\widehat{\mathbb{K}}|$ due to the random choice of $\{\widehat{\beta}_j\}$'s. Roughly, this holds because for the index $j \in \{1, \ldots, M\}$ at which $\mathcal{A}$ uses a non-zero additive value $d \in \widehat{\mathbb{K}} \setminus \{0\}$ in the invocation of $\widehat{\mathcal{F}}_{\text{mult}}$ for computing $[\![\widehat{r} \cdot v_j]\!]$, we have $\mathsf{val}([\![\widehat{r} \cdot v_j]\!])_H = \widehat{r} \cdot v_j + d$. Then, $\mathsf{val}([\![\widehat{T}]\!])_H$ contains $\widehat{\beta}_j \cdot d$ as a monomial, which is uniformly distributed over $\widehat{\mathbb{K}}$ since so is $\widehat{\beta}_j$. Thus, $\mathsf{val}([\![\widehat{T}]\!])_H$ can be zero only with probability $1/|\widehat{\mathbb{K}}|$. The remaining cases can be shown with similar arguments, and the probability that $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ occurring in Case (2) is bounded by $2/|\widehat{\mathbb{K}}|$ due to the random choices of $\widehat{\alpha}_k$'s and $\widehat{r}$, and that in Case (3) is bounded by $1/|\widehat{\mathbb{K}}|$ due to the random choice of $\widehat{r}$.

[**Proof**] Note that during the execution of the protocol, $\mathcal{A}$ may mound additive attacks when calling $\mathcal{F}_{\text{mult}}$ and $\widehat{\mathcal{F}}_{\text{mult}}$ at the computation of multiplication gates in Step 4, and calling $\widehat{\mathcal{F}}_{\text{product}}$ at (c) in Step 5.

Regarding (1), for each $j \in \{1, \ldots, M\}$, let $\widehat{a}_j \in \widehat{\mathbb{K}}$ denote the added value in $\mathcal{A}$'s additive attack in the invocation of $\widehat{\mathcal{F}}_{\text{mult}}$ for the randomized share $[\![\widehat{r} \cdot v_j]\!]$ for the j-th input wire. Then, we have $\mathsf{val}([\![\widehat{r} \cdot v_j]\!])_H = \widehat{r} \cdot v_j + \widehat{a}_j$.

Regarding (2), consider the computation at the k-th multiplication gate for each $k \in \{1, \ldots, N\}$. Parties initially hold pairs $([x_k], [\![\widehat{r} \cdot x_k]\!])$ and $([y_k], [\![\widehat{r} \cdot y_k]\!])$ that correspond to the left and right wires, respectively. Here, $[x_k]$ and $[\![\widehat{r} \cdot x_k]\!]$ may not have been computed correctly (i.e. it is possible that $\mathsf{val}([\![\widehat{r} \cdot x_k]\!])_H \neq \widehat{r} \cdot x_k$), since $\mathcal{A}$ might have mounted additive attacks in previous multiplication gates and in the computation of input wires. However, for proving the lemma, only their relative relation taking into account all "accumulated" errors so far matters. That is, we have $\mathsf{val}([\![\widehat{r} \cdot x_k]\!])_H = \widehat{r} \cdot x_k + \widehat{b}_k$, where $\widehat{b}_k \in \widehat{\mathbb{K}}$ denotes the accumulated errors.

Then, we denote by $[z_k]$ the shares returned from $\mathcal{F}_{\text{mult}}$ on inputs $[x_k]$ and $[y_k]$. Also, we denote by $[\![\widehat{r} \cdot z_k]\!]$ the shares returned from $\widehat{\mathcal{F}}_{\text{mult}}$ on inputs $[\![\widehat{r} \cdot x_k]\!]$ and $[\![y_k]\!]$. Letting $c_k \in \mathbb{K}$ (resp. $\widehat{d}_k \in \widehat{\mathbb{K}}$) denote the additive value in $\mathcal{A}$'s additive attack for the invocation of $\mathcal{F}_{\text{mult}}$ (resp. $\widehat{\mathcal{F}}_{\text{mult}}$), we have $\mathsf{val}([z_k])_H = x_k \cdot y_k + c_k$ and $\mathsf{val}([\![\widehat{r} \cdot z_k]\!])_H = (\widehat{r} \cdot x_k + \widehat{b}_k) \cdot y_k + \widehat{d}_k$.

Regarding (3), letting $\widehat{f}, \widehat{g} \in \widehat{\mathbb{K}}$ denote the additive values in $\mathcal{A}$'s additive attacks in the invocations of $\widehat{\mathcal{F}}_{\text{product}}$, and using the above introduced $\{\mathsf{val}([\![\widehat{r} \cdot v_m]\!])_H\}_{m=1}^{M}$, $\{\mathsf{val}([z_k])_H\}_{k=1}^{N}$, and $\{\mathsf{val}([\![\widehat{r} \cdot z_k]\!])_H\}_{k=1}^{N}$, we have

$$\mathsf{val}([\![\widehat{u}]\!])_H = \sum_{k=1}^{N} \widehat{\alpha}_k \cdot \left(\left(\widehat{r} \cdot x_k + \widehat{b}_k \right) \cdot y_k + \widehat{d}_k \right) + \sum_{j=1}^{M} \widehat{\beta}_j \cdot (\widehat{r} \cdot v_j + \widehat{a}_j) + \widehat{f},$$

$$\mathsf{val}([\![\widehat{w}]\!])_H = \sum_{k=1}^{N} \widehat{\alpha}_k \cdot (x_k \cdot y_k + c_k) + \sum_{j=1}^{M} \widehat{\beta}_j \cdot v_j + \widehat{g}.$$

Thus, we have

$$\begin{aligned}
\mathsf{val}([\![\widehat{T}]\!])_H &= \mathsf{val}([\![\widehat{u}]\!])_H - \widehat{r} \cdot \mathsf{val}([\![\widehat{w}]\!])_H \\
&= \sum_{k=1}^{N} \widehat{\alpha}_k \cdot \left(\left(\widehat{r} \cdot x_k + \widehat{b}_k \right) \cdot y_k + \widehat{d}_k \right) + \sum_{j=1}^{M} \widehat{\beta}_j \cdot (\widehat{r} \cdot v_j + \widehat{a}_j) + \widehat{f} \\
&\quad - \widehat{r} \cdot \left(\sum_{k=1}^{N} \widehat{\alpha}_k \cdot (x_k \cdot y_k + c_k) + \sum_{j=1}^{M} \widehat{\beta}_j \cdot v_j + \widehat{g} \right) \\
&= \sum_{k=1}^{N} \widehat{\alpha}_k \cdot \left(\widehat{b}_k \cdot y_k + \widehat{d}_k - \widehat{r} \cdot c_k \right) + \sum_{j=1}^{M} \widehat{\beta}_j \cdot \widehat{a}_j + \widehat{f} - \widehat{r} \cdot \widehat{g}.
\end{aligned}$$

In the following, we show that the probability that $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ holds is at most $2/|\widehat{\mathbb{K}}|$ by the case classification on which stage $\mathcal{A}$ submits a non-zero additive value for the first time: Step 3, Step 4, or Step 5.

- Case 1 – Step 3: In this case, there exists some $j \in \{1, \dots, M\}$ such that $\widehat{a}_j \neq 0$: Let j_0 be the smallest such j for which this holds. In this case, $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ occurs if and only if

$$\widehat{\beta}_{j_0} = \left(- \sum_{k=1}^{N} \widehat{\alpha}_k \cdot \left(\widehat{b}_k \cdot y_k + \widehat{d}_k - \widehat{r} \cdot c_k \right) - \sum_{\substack{j=1 \\ j \neq j_0}}^{M} \widehat{\beta}_j \cdot \widehat{a}_j - \widehat{f} + \widehat{r} \cdot \widehat{g} \right) \cdot \frac{1}{\widehat{a}_{j_0}}.$$

  This holds with probability $1/|\widehat{\mathbb{K}}|$ since $\widehat{\beta}_{j_0} \in \widehat{\mathbb{K}}$ is distributed uniformly, and chosen independently of other values.
- Case 2 – Step 4: In this case, $\widehat{a}_j = 0$ for all $j \in \{1, \dots, M\}$ and there exists some $k \in \{1, \dots, N\}$ such that $c_k \neq 0$ or $\widehat{d}_k \neq 0$. Let k_0 be the smallest such k for which this holds. Since the invocation of $\mathcal{F}_{\mathrm{mult}}$ or $\widehat{\mathcal{F}}_{\mathrm{mult}}$ for the k_0-th multiplication gate is the first time $\mathcal{A}$ uses a non-zero value for its additive attack, the "accumulated" error at this point is also zero, i.e. we have $\widehat{b}_{k_0} = 0$. Thus, $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ occurs if and only if

$$\widehat{\alpha}_{k_0} \cdot \left(\widehat{d}_{k_0} - \widehat{r} \cdot c_{k_0} \right) = - \sum_{\substack{k=1 \\ k \neq k_0}}^{N} \widehat{\alpha}_k \cdot \left(\widehat{b}_k \cdot y_k + \widehat{d}_k - \widehat{r} \cdot c_k \right) - \widehat{f} + \widehat{r} \cdot \widehat{g}.$$

  If $\widehat{d}_{k_0} - \widehat{r} \cdot c_{k_0} \neq 0$, then the above equality holds with probability $1/|\widehat{\mathbb{K}}|$ due to the fact that $\widehat{\alpha}_{k_0} \in \widehat{\mathbb{K}}$ is distributed uniformly, and chosen independently of other values. On the other hand, $\widehat{d}_{k_0} - \widehat{r} \cdot c_{k_0} = 0$ occurs with probability at most $1/|\widehat{\mathbb{K}}|$ since $\widehat{r} \in \widehat{\mathbb{K}}$ is distributed uniformly, and is completely unknown to $\mathcal{A}$ at the timing the computation for k_0-th multiplication gate is executed thanks to the perfect security of the underlying secret sharing scheme and $\widehat{\mathcal{F}}_{\mathrm{rand}}$. Putting it together, the probability that the above equation holds is at most $\frac{1}{|\widehat{\mathbb{K}}|} + \left(1 - \frac{1}{|\widehat{\mathbb{K}}|}\right) \cdot \frac{1}{|\widehat{\mathbb{K}}|} < \frac{2}{|\widehat{\mathbb{K}}|}$.

– Case 3 – Step 5: In this case, $\widehat{a}_j = 0$ for all $j \in \{1, \ldots, M\}$ and $\widehat{b}_k, c_k, \widehat{d}_k = 0$ for all $k \in \{1, \ldots, N\}$. (Note that $\widehat{b}_k$'s are zero since no accumuated errors exist.) By the assumption of the lemma, $\mathcal{A}$ uses at least one non-zero additive value, and thus either $\widehat{f} \neq 0$ or $\widehat{g} \neq 0$ holds. Then, $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ occurs if and only if $\widehat{f} - \widehat{r} \cdot \widehat{g} = 0$, which occurs with probability at most $1/|\widehat{\mathbb{K}}|$ again due to the perfect security of the underlying secret sharing scheme and $\widehat{\mathcal{F}}_{\text{rand}}$.

In all the cases, the probability that $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ occurs is at most $2/|\widehat{\mathbb{K}}|$. ■

Armed with the above key lemma, we now give the proof of Theorem 4.16.

[**Proof**] (of Theorem 4.16) Let $\mathcal{A}$ be the real adversary who controls the set of corrupted parties $\mathcal{C}$. The simulator $\mathcal{S}$ for $\mathcal{A}$ works as follows.

Initially, $\mathcal{S}$ receives from $\mathcal{A}$ the set of corrupted parties' inputs $\{v_i\}_{i \in \mathcal{C}}$ and input shares $\{[v_j]_{\mathcal{C}}\}_{j=1}^{M}$ that $\mathcal{A}$ sends to $\mathcal{F}_{\text{input}}$. For each $j \in H$, $\mathcal{S}$ computes $[v_j] = ([v_j]_1, \ldots, [v_j]_n) \leftarrow \mathsf{Share}(0, [v_j]_{\mathcal{C}})$ (i.e. shares of 0), which is used for simulating the honest parties' behaviors throughout the simulation for $\mathcal{A}$.

After this step, direct communication among the honest parties and the corrupted parties controled by $\mathcal{A}$ occur only from (c) iii. of *Verificaiton stage* (Step 5). Until then, only local computations or invocations of functionalities of $\widehat{\mathcal{F}}_{\text{rand}}$, $\mathcal{F}_{\text{mult}}$, $\widehat{\mathcal{F}}_{\text{mult}}$, and $\widehat{\mathcal{F}}_{\text{product}}$ are performed. For each invocation of $\widehat{\mathcal{F}}_{\text{rand}}$, $\mathcal{S}$ just receives from $\mathcal{A}$ the corrupted parties' shares. For each invocation of $\mathcal{F}_{\text{mult}}$ for input shares $[x]$ and $[y]$, $\mathcal{S}$ hands $\mathcal{A}$ the corrupted parties' shares[4] $[x]_{\mathcal{C}}$ and $[y]_{\mathcal{C}}$, and receives from $\mathcal{A}$ the additive value d (for an additive attack) and the corrupted parties resulting shares $[\![x \cdot y]\!]_{\mathcal{C}}$ (which may not be correctly computed). The invocations of $\widehat{\mathcal{F}}_{\text{mult}}$ and $\widehat{\mathcal{F}}_{\text{product}}$ are simulated similarly.

When the simulated execution of the protocol reaches (c) iii. of *Verification stage*, $\mathcal{S}$ proceeds as follows: $\mathcal{S}$ chooses $\widehat{r} \in \widehat{\mathbb{K}}$ uniformly at random, and generates $[\![\widehat{r}]\!] = ([\![\widehat{r}]\!]_1, \ldots, [\![\widehat{r}]\!]_n) \leftarrow \widehat{\mathsf{Share}}(\widehat{r}, [\![\widehat{r}]\!]_{\mathcal{C}})$, where $[\![\widehat{r}]\!]_{\mathcal{C}}$ denotes the corrupted parties's shares that $\mathcal{S}$ received from $\mathcal{A}$ when simulating Step 2. Then, $\mathcal{S}$ simulates the opening of $\widehat{r}$ by handing $\mathcal{A}$ all of the honest parties' shares $[\![\widehat{r}]\!]_H$. If any of the honest parties aborts (which $\mathcal{S}$ can know from the shares $[\![\widehat{r}]\!]$), then $\mathcal{S}$ sends $\perp$ to all parties, externally sends the abort message to the trusted third party computing $\mathcal{F}$, and halts.

Next, $\mathcal{S}$ simulates $\widehat{\mathcal{F}}_{\text{checkZero}}$ as follows: If $\mathcal{A}$ has used some non-zero additive value in the invocations of $\mathcal{F}_{\text{mult}}$, $\widehat{\mathcal{F}}_{\text{mult}}$, or $\widehat{\mathcal{F}}_{\text{product}}$, then $\mathcal{S}$ sends *reject* to $\mathcal{A}$, externally sends the abort message to the trusted party computing $\mathcal{F}$, and halts. Otherwise (i.e. no non-zero additive value was used), $\mathcal{S}$ sends 0 to $\mathcal{A}$. If $\mathcal{S}$ receives *reject* back from $\mathcal{A}$, $\mathcal{S}$ does the same as above. If $\mathcal{S}$ receives *accept* back from $\mathcal{A}$, $\mathcal{A}$ proceeds to the next step.

If no abort had occurred, $\mathcal{S}$ proceeds to *Output reconstruction* (Step 6). First, $\mathcal{S}$ externally sends the corrupted parties' inputs $\{[v_j]_{\mathcal{C}}\}_{j=1}^{M}$ to the trusted party for computing $\mathcal{F}$, and receives the output values for each output wire of the circuit C that are associated with the corrupted parties. Then, $\mathcal{S}$ simulates the honest parties in the reconstruction of the corrupted parties' outputs by computing the honest parties' shares of each output wire of the circuit C, using the corrupted parties' shares on the output wire and the actual output values that $\mathcal{S}$ received from the trusted party. In addition, $\mathcal{S}$ receives the messages sent from $\mathcal{A}$ to the honest parties for the reconstructions. If any of the messages in the reconstruction of an output wire of the circuit C associated with an honest party P_i is incorrect (i.e. the shares sent by $\mathcal{A}$ are not the correct shares), then $\mathcal{S}$ sends abort_i to instruct the trusted party to not send the output to P_i. Otherwise, $\mathcal{S}$ sends $\mathsf{continue}_i$ to the trusted party, instructing it to send the output to P_i.

The above completes the description of $\mathcal{S}$. Due to the security of the underlying secret sharing scheme, the distributions of the input shares that $\mathcal{S}$ initially compute for honest parties are perfectly indistinguishable from the actual ones. Furthermore, $\mathcal{S}$'s simulation of the functionalities $\mathcal{F}_{\text{input}}$, $\widehat{\mathcal{F}}_{\text{rand}}$, $\mathcal{F}_{\text{mult}}$, $\widehat{\mathcal{F}}_{\text{mult}}$, and $\widehat{\mathcal{F}}_{\text{product}}$ are perfect. Moreover, unless $\mathcal{S}$ fails to simulate $\widehat{\mathcal{F}}_{\text{checkZero}}$, which we will show occurs with probability at most $3/|\widehat{\mathbb{K}}|$, $\mathcal{S}$'s simulation of the remaining procedures of $\mathcal{S}$ for computing the outputs is also perfect.

[4] Throughout the simulation, $\mathcal{S}$ knows the corrupted parties' shares on the input wires of the gate being computed, and thus can conduct this step.

Hence, the distribution of $\mathcal{A}$'s view is identical to that in the real execution except with probability at most $3/|\widehat{\mathbb{K}}| \le 2^{-\kappa}$.

We now show that $\mathcal{S}$'s simulation of $\widehat{\mathcal{F}}_{\mathrm{checkZero}}$ fails with probability at most $3/|\widehat{\mathbb{K}}|$.

– If $\mathcal{A}$ has not used any non-zero additive value for the invocations of $\mathcal{F}_{\mathrm{mult}}$, $\widehat{\mathcal{F}}_{\mathrm{mult}}$, or $\widehat{\mathcal{F}}_{\mathrm{product}}$, then $\mathcal{S}$ sends 0 to $\mathcal{A}$ and outputs *accept* or *reject* to the honest parties depending on $\mathcal{A}$'s reply. If $\mathcal{A}$ does so in the real protocol execution, we always have $\widehat{T} = 0$, in which case $\widehat{\mathcal{F}}_{\mathrm{checkZero}}$ sends 0 to $\mathcal{A}$, and proceeds in exactly the same way as $\mathcal{S}$ does for $\mathcal{A}$. Thus, in this case $\mathcal{S}$'s simulation of $\widehat{\mathcal{F}}_{\mathrm{checkZero}}$ is perfect.
– Otherwise, i.e. if $\mathcal{A}$ has used some non-zero value for its additive attacks, $\mathcal{S}$ always outputs *reject*. On the other hand, if $\mathcal{A}$ does so in the real protocol execution, $\widehat{\mathcal{F}}_{\mathrm{checkZero}}$ may output *accept* if either $\widehat{T} = 0$, or $\widehat{T} \neq 0$ but it chose *accept* with probability $1/|\widehat{\mathbb{K}}|$. By Lemma 4.17, the probability that $\mathsf{val}([\![\widehat{T}]\!])_H = 0$ occurs in such a situation is less than $2/|\widehat{\mathbb{K}}|$. Thus, $\widehat{\mathcal{F}}_{\mathrm{checkZero}}$ outputs *accept* with probability at most $\frac{2}{|\widehat{\mathbb{K}}|} + \left(1 - \frac{2}{|\widehat{\mathbb{K}}|}\right) \cdot \frac{1}{|\widehat{\mathbb{K}}|} < \frac{3}{|\widehat{\mathbb{K}}|}$. This implies that the probability of simulation error in this case is also bounded by $3/|\widehat{\mathbb{K}}|$.

As seen above, $\mathcal{S}$ succeeds in simulating $\widehat{\mathcal{F}}_{\mathrm{checkZero}}$ except with probability at most $3/|\widehat{\mathbb{K}}|$. This completes the proof. ■

Acknowledgement

A part of this work was supported by JST CREST grantnumber JPMJCR19F6.

References

1. T. Araki, A. Barak, J. Furukawa, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *IEEE Symposium on Security and Privacy, SP 2017*, 2017.
2. T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *ACM CCS 2016*, pages 805–817, 2016.
3. D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991.
4. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC 2008*, pages 213–230, 2008.
5. A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution.* PhD thesis, Israel Institute of Technology, 1996.
6. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC 1988*, pages 1–10, 1988.
7. E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *CRYPTO 2012*, pages 663–680, 2012.
8. R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145, 2001.
10. I. Cascudo, R. Cramer, C. Xing, and C. Yuan. Amortized complexity of information-theoretically secure MPC revisited. In *CRYPTO 2018, Part III*, pages 395–426, 2018.
11. K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *CRYPTO 2018, Part III*, pages 34–64, 2018.
12. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *TCC 2005*, pages 342–362, 2005.
13. I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO 2007*, pages 572–590, 2007.
14. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO 2012*, pages 643–662, 2012.
15. J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT 2017, Part II*, pages 225–255, 2017.

16. D. Genkin, Y. Ishai, and A. Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In *CRYPTO 2015, Part II*, pages 721–741, 2015.
17. D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, and E. Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC 2014*, pages 495–504, 2014.
18. D. Genkin, Y. Ishai, and M. Weiss. Binary AMD circuits from secure multiparty computation. In *TCC 2016-B,*, pages 336–366, 2016.
19. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.
20. S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *J. Cryptology*, 18(3):247–287, 2005.
21. M. Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting.* PhD thesis, ETH Zurich, 2001.
22. M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Globecom 1987*, pages 99–102, 1987.
23. R. Kikuchi, K. Chida, D. Ikarashi, and K. Hamada. Password-based authentication protocol for secret-sharing-based multiparty computation. *IEICE Transactions*, 101-A(1):51–63, 2018.
24. E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.
25. Y. Lindell and A. Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In *ACM CCS 2017*, pages 259–276, 2017.
26. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Y. Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.
27. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

A Malicious Security in Secure Computation

Most of the contents in this section are taken verbatim from [25, Appendix A]. In the following, the security parameter is denoted λ; negligible functions and computational indistinguishability are defined in the standard way, with respect to non-uniform polynomial-time distinguishers.

Ideal versus Real Model Definition. We use the ideal/real simulation paradigm in order to define security, where an execution in the real model is compared to an execution in the ideal model where an incorruptible trusted party computes the functionality for the parties [8,19]. We define *security with abort* (and without fairness), meaning that the corrupted parties may receive output while the honest parties do not. Our definition does *not* guarantee *unanimous abort*, meaning that some honest party may receive output while the other does not. It is easy to modify our protocols so that the honest parties unanimously abort by running a single (weak) Byzantine agreement at the end of the execution [20]; we therefore omit this step for simplicity.

The Real Model. In the real model, an n-party protocol π is executed by the parties. For simplicity, we consider a synchronous network that proceeds in rounds and a rushing adversary, meaning that the adversary receives its incoming messages in a round before it sends its outgoing message. The adversary $\mathcal{A}$ can be active; it sends all messages in place of the corrupted parties, and can follow any arbitrary strategy. The honest parties follow the instructions of the protocol.

Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary who can control $t < \frac{n}{2}$ parties. Let $Real_{\pi,\mathcal{A}(z),I}(x_1,\ldots,x_n,\lambda)$ denote the output of the honest parties and $\mathcal{A}$ in the real execution of π, with inputs $x_1,\ldots,x_n$, auxiliary-input z for $\mathcal{A}$, and security parameter λ.

The Ideal Model. We define the ideal model, for any (possibly reactive) functionality $\mathcal{F}$, receiving inputs from $P_1,\ldots,P_n$ and providing them with outputs. Let $I \subset \{1,\ldots,n\}$ be the set of indices of the corrupted parties controlled by the adversary. The ideal execution proceeds as follows:

- **Send inputs to the trusted party:** Each honest party P_j sends its specified input x_j to the trusted party. A corrupted party P_i controlled by the adversary may either send its specified input x_i, some other x'_i or an abort message.
- **Early abort option:** If the trusted party received abort from the adversary $\mathcal{A}$, it sends $\perp$ to all parties and terminates. Otherwise, it proceeds to the next step.
- **Trusted party sends output to the adversary:** The trusted party computes each party's output as specified by the functionality $\mathcal{F}$ based on the inputs received; denote the output of P_j by y_j. The trusted party then sends $\{y_i\}_{i\in I}$ to the corrupted parties.
- **Adversary instructs trusted party to continue or halt:** For each $j \in \{0, \ldots, n-1\}$ with $j \notin I$, the adversary sends the trusted party either abort_j or $\mathsf{continue}_j$. For each $j \notin I$:
  - If the trusted party received abort_j then it sends P_j the abort value $\perp$ for output.
  - If the trusted party received $\mathsf{continue}_j$ then it sends P_j its output value y_j.
- **Outputs:** The honest parties always output the output value they obtained from the trusted party, and the corrupted parties outputs nothing.

Let $\mathcal{S}$ be a non-uniform probabilistic polynomial-time adversary controlling parties P_i for $i \in I$. Let $Ideal_{\mathcal{F},\mathcal{S}(z),I}(x_1, \ldots, x_n, \lambda)$ denote the output of the honest parties and $\mathcal{S}$ in an ideal execution with the functionality $\mathcal{F}$, inputs $x_1, \ldots, x_n$ to the parties, auxiliary-input z to $\mathcal{S}$, and security parameter λ.

Security. Informally speaking, the definition says that protocol π securely computes a functionality $\mathcal{F}$ if adversaries in the ideal model can simulate executions of the real model protocol. In some of our protocols there is a statistical error that is not dependent on the computational security parameter. As in [26], we formalize security in this model by saying that the distinguisher can distinguish with probability at most this error *plus* some factor that is negligible in the security parameter. This is formally different from the standard definition of security since the statistical error does not decrease as the security parameter increases.

Definition A.1. *Let $\mathcal{F}$ be an n-party functionality, and let π be an n-party protocol. We say that π se-*curely computes $\mathcal{F}$ with abort in the presence of an adversary controlling $t < \frac{n}{2}$ parties, *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time simulator/adversary $\mathcal{S}$ in the ideal model with $\mathcal{F}$, such that for every $I \subset \{1, \ldots, n\}$,*

$$\left\{Ideal_{F,\mathcal{S}(z),I}(x_1, \ldots, x_n, \lambda)\right\} \overset{c}{\approx} \left\{Real_{\pi,\mathcal{A}(z),I}(x_1, \ldots, x_n, \lambda)\right\}$$

where $x_1, \ldots, x_n \in \{0,1\}^*$ under the constraint that $|x_1| = \cdots = |x_n|$, $z \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$. We say that π securely computes $\mathcal{F}$ with abort in the presence of t active parties with statistical error $2^{-\kappa}$ *if there exists a negligible function $\mu(\cdot)$ such that the distinguishing advantage of the adversary is less than $2^{-\kappa} + \mu(\lambda)$.*

Hybrid Model. We prove the security of our protocols in a hybrid model, where parties run a protocol with real messages and also have access to a trusted party computing a subfunctionality for them. The modular sequential composition theorem of [8] states that one can replace the trusted party computing the subfunctionality with a real secure protocol computing the subfunctionality. When the subfunctionality is g, we say that the protocol works in the *g-hybrid model.*

Universal Composability [9]. Protocols that are proven secure in the universal composability framework have the property that they maintain their security when run in parallel and concurrently with other secure and insecure protocols. In [24, Theorem 1.5], it was shown that any protocol that is proven secure with a black-box non-rewinding simulator and also has the property that the inputs of all parties are fixed before the execution begins (called input availability or start synchronization in [24]), is also secure under universal composability. Since the input availability property holds for all of our protocols and subprotocols, it is sufficient to prove security in the classic stand-alone setting and automatically derive universal composability from [24]. We remark that this also enables us to call the protocol and subprotocols that we use in parallel and concurrently (and not just sequentially), enabling us to achieve more efficient computation (e.g., by running many executions in parallel or running each layer of a circuit in parallel).